Efficient verification of network fault tolerance via counterexample-guided refinement*

Nick Giannarakis[1], Ryan Beckett[2], Ratul Mahajan[3,4], and David Walker[1]

[1] Princeton University, Princeton NJ 08544, USA
{ng8,dpw}@cs.princeton.edu
[2] Microsoft Research, Redmond WA 98052, USA
ryan.beckett@microsoft.com
[3] University of Washington, Seattle WA 98195, USA
[4] Intentionet, Seattle WA, USA
ratul@cs.washington.edu

Abstract. We show how to verify that large data center networks satisfy key properties such as all-pairs reachability under a bounded number of faults. To scale the analysis, we develop algorithms that identify network symmetries and compute small abstract networks from large concrete ones. Using counter-example guided abstraction refinement, we successively refine the computed abstractions until the given property may be verified. The soundness of our approach relies on a novel notion of network approximation: routing paths in the concrete network are not precisely simulated by those in the abstract network but are guaranteed to be "at least as good." We implement our algorithms in a tool called Origami and use them to verify reachability under faults for standard data center topologies. We find that Origami computes abstract networks with 1–3 orders of magnitude fewer edges, which makes it possible to verify large networks that are out of reach of existing techniques.

1 Introduction

Most networks decide how to route packets from point A to B by executing one or more distributed routing protocols such as the Border Gateway Protocol (BGP) and Open Shortest Path First (OSPF). To achieve end-to-end policy objectives related to cost, load balancing, security, etc., network operators author configurations for each router. These configurations control various aspects of the route computation such as filtering and ranking route information received from neighbors, information injection from one protocol to another, and so on.

This flexibility, however, comes at a cost: Configuring individual routers to enforce the desired policies of the distributed system is complex and error-prone [16,22]. The problem of configuration is further compounded by three challenges. The first is network scale. Large networks such as those of cloud

* This work was supported in part by NSF Grants 1703493 and 1837030, and gifts from Cisco and Facebook. Any opinions, findings, and conclusions expressed are those of the authors and do not necessarily reflect those of the NSF, Cisco or Facebook.

providers can consist of millions of lines of configuration spread across thousands of devices. The second is that operators must account for the interaction with external neighbors who may sent arbitrary routing messages. Finally one has to deal with *failures*. Hardware failures are common [15] and lead to a combinatorial explosion of different possible network behaviors.

To combat the complexity of distributed routing configurations, researchers have suggested a wide range of network verification [2,14,26] and simulation [11,12,24] techniques. These techniques are effective on small and medium-sized networks, but they cannot analyze data centers with 1000s of routers and all their possible failures. To enable scalable analyses, it seems necessary to exploit the symmetries that exist in most large real networks. Indeed, other researchers have exploited symmetries to scale verification in the past [3,23]. However, it has never been possible to account for failures, as they introduce asymmetries that change routing behaviors in unpredictable ways.

To address this challenge, we develop a new algorithm for verifying reachability in networks in the presence of faults, based on the idea of counterexample-guided abstraction refinement (CEGAR) [5]. The algorithm starts by factoring out symmetries using techniques developed in prior work [3] and then attempts verification of the abstract network using an SMT solver. If verification succeeds, we are done. However, if verification fails, we examine the counter-example to decide whether we have a true failure or we must refine the network further and attempt verification anew. By focusing on reachability, the refinement procedure can be accelerated by using efficient graph algorithms, such as min cut, to rule out invalid abstractions in the middle of the CEGAR loop.

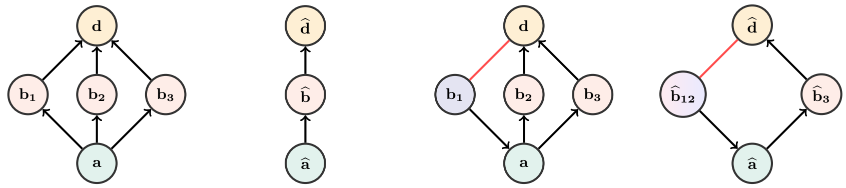
We prove the correctness of our algorithm using a new theory of faulty networks that accounts for the impact of all combinations of k failures. Our key insight is that, while routes computed in the abstract network may not simulate those of the concrete network exactly, under the right conditions they are guaranteed to *approximate* them. The approximation relation between concrete and abstract networks suffices to verify key properties such as reachability.

We implemented our algorithms in a tool called Origami and measured their performance on common data center network topologies. We find that Origami computes abstract networks with 1-3 orders of magnitude fewer edges. This reduction speeds verification dramatically and enables verification of networks that are out of reach of current state-of-the-art tools [2].

2 Key Ideas

The goal of Origami is to speed up network verification in the presence of faults, and it does so by computing small, abstract networks with *similar* behavior to a given concrete network.

As a first approximation, one can view a network as a directed graph capturing the physical topology, and its routing solution as a subgraph where the remaining edges denote the forwarding decision at each node for some fixed destination. In the absence of faults, given a concrete and abstract network, one



(a) (b) (c) (d)

Fig. 1: All graph edges shown correspond to edges in the network topology, and we draw edges as directed to denote the direction of forwarding eventually determined for each node by the distributed routing protocols for a fixed destination d. In (a) nodes use shortest path routing to route to the destination d. (b) shows a compressed network that precisely captures the forwarding behavior of (a). Figure (c) shows how forwarding is impacted by a link failure, shown as a red line. Figure (d) shows a compressed network that is sound approximation of the original network for any single link failure.

can define a natural notion of similarity as a graph homomorphism: assigning each concrete node a corresponding abstract node such that, for any solution to the routing problem, the concrete node forwards “in the same direction” as the corresponding abstract node. For example, the concrete network in Figure 1a is related to its abstract counterpart in Figure 1b according to the node colors.

To illustrate, consider the network of Figure 1a, which computes routes towards the destination d via a distributed shortest path routing protocol. Nodes b_1, b_2, and b_3 all play the same “role” and we can exploit the symmetries in their behavior to construct the compressed network presented in Figure 1b. In the latter diagram, each of b_1, b_2, b_3 may be assigned to $\widehat{b}$ (and d to $\widehat{d}$), as all these nodes “route in the same direction:”. b_1, b_2, b_3 route to concrete node d and $\widehat{b}$ routes to the corresponding node $\widehat{d}$. Moreover, the compressed network preserves many interesting properties of the concrete one, such as the existence of a path from any node to the destination, the length of such a path, and the set of nodes traversed (modulo the abstraction function). In fact, compressing the network in this way is the approach taken by Bonsai [3], which works well in cases where many symmetries exist. The advantage of using compression is that a compressed network is often dramatically faster to verify than the original network.

Unfortunately, we run into two significant problems when defining abstractions in this manner in the presence of faults. First, the concrete nodes of Figure 1a have at least 2 disjoint paths to the destination whereas abstract nodes of Figure 1b have just one path to the destination, so the abstract network does not preserve the desired fault tolerance properties. Second, consider Figure 1c, which illustrates how the routing decisions change when a failure occurs. Here, the nodes (b_1 in particular) no longer route “in the same direction” as the original network or its abstraction. Hence the invariant connecting concrete and abstract networks is violated.

Returning to the previous example, consider Figure 1c, which presents the routing behavior of the network when the link between b_1 and d has failed. The nodes $\{b_1, b_2, b_3\}$ no longer have identical routing behaviors, and instead exhibit two different behaviors. In this scenario, the behavior of b_1 differs from the behavior of b_2, b_3 but in a setting where the link between b_2 and d is the one that fails it will be b_2 that has a different behavior. Hence it is not possible to statically (*i.e.*, when compressing the network) determine which b_i nodes exhibit the same routing behavior. Even worse, in general statically determining how many routing behaviors nodes in each abstract "role" may exhibit under arbitrary (but bounded in number) link failures seems to require solving the very routing problem we are trying to simplify. The bottom line is that given our current definition of similarity (routes "in the same direction") abstraction of any pair of concrete nodes as a single abstract node would appear impossible. There is always the potential for an ill-placed failure to generate a difference. To develop a sound compression algorithm in the presence of failures, we must go back to the drawing board.

Lossy compression.

To achieve compression given a bounded number of link failures, we relax the notion of similarity between concrete and abstract nodes: A node in the abstract network may merely *approximate* the behavior of concrete nodes. This makes it possible to compress nodes that, in the presence of failures, may route differently. In general, when we fail a single link in the abstract network, we are over-approximating the failures in the concrete network by failing multiple concrete links, possibly more than desired. Nevertheless, the paths taken in the concrete network can only deviate so much from the paths found in the abstract network:

Property 1. If a node has a route to the destination in the presence of k link failures then it has a route that is "at least as good" (as prescribed by the routing protocol) in the presence of k' link failures for $k' < k$.

In general, the *preference* of routes (*"at least as good"*) is protocol dependent. For instance, in the case of shortest path routing, "at least as good" is determined by the path length. In the case of shortest path routing, one path is "at least as good" as another when the length of the first path is less than or equal to the length of the second path. This relation suffices to verify important network reliability properties, such as reachability, in the presence of faults. Just as importantly, it allows us to achieve effective network compression to scale verification.

Revisiting our example, consider the new abstract network of Figure 1d. When the link between $\widehat{b}_{12}$ and $\widehat{d}$ has failed, $\widehat{b}_{12}$ still captures the behavior of b_1 precisely. However, b_2 has a better (in this case better means shorter) path to d. Despite this difference, if the operator's goal was to prove reachability to the destination under any single fault, then this abstract network suffices.

To summarize so far, we can observe that the abstract network in **??** has several nice properties:

1. Using a verification tool (e.g., Minesweeper [2]) we can verify that all nodes in this abstract network can reach the destination despite any single link failure.

2. The abstract network is related to the concrete network by virtue of the fact that concrete nodes have paths "at least as good" as their corresponding abstract nodes.

By 1 and 2, we can conclude that all *concrete* nodes have a path to the destination, despite the presence of any single fault, and without having had to incur the expense of verifying this property of the large (in practice, much larger) concrete network.

From specification to algorithm. It is not too difficult to find abstract networks that approximate a concrete network; the challenge is finding a valid abstract network that is *small enough* to make verification feasible and yet *large enough* to include sufficiently many paths to verify the given fault tolerance property. Rather than attempting to compute a single abstract network with the right properties all in one shot, we search the space of abstract networks using an algorithm based on *counter-example guided abstraction refinement* [5].

The CEGAR algorithm begins by computing the smallest possible valid abstract network. In the example above, this corresponds to the original compressed network in Figure 1b, which faithfully approximates the original network when there are no link failures. However, if we try to verify reachability in the presence of a single fault, we will conclude that nodes $\widehat{b}$ and $\widehat{a}$ have no route to the destination when the link between $\widehat{b}$ and $\widehat{d}$ fails. The counterexample due to this failure could of course be spurious (and indeed it is). Fortunately, we can easily distinguish whether such a failure is due to lack of connectivity or an artifact of over-abstracting, by calculating the number of corresponding concrete failures. In this example a failure on the link $\langle \widehat{b}, \widehat{d} \rangle$ corresponds to 3 concrete failures. Since we are interested in verifying reachability for a single failure this cannot constitute an actual counterexample.

The next step is to *refine* our abstraction by splitting some of the abstract nodes. The idea is to use the counterexample from the previous iteration to split the abstract network in a way that avoids giving rise to the same spurious counterexample in the next iteration (Section 5). Doing so results in the somewhat larger network of Figure 1d. A second verification pass over this larger network takes longer, but succeeds.

3 The Network Model

Though there are a wide variety of routing protocols in use today, they share a lot in common. Griffin *et al.* [17] showed that protocols like BGP and others solve instances of the *stable paths problem*, a generalization of the shortest paths problem, and Sobrinho [25] demonstrated their semantics and properties can be modelled using routing algebras. We extend these foundations by defining *stable paths problems with faults* (SPPFs), an extension of the classic Stable Paths Problem that admits the possibility of a bounded number of link failures. In later sections, we use this network model to develop generic network compression algorithms and reason about their correctness.

Stable path problems with faults (SPPFs): An SPPF is an instance of the stable paths problem with faults. Informally, each instance defines the routing behavior of an operational network. The definition includes both the network topology as well as the routing policy. The policy specifies the way routing messages are transformed as they travel along links and through the user-configured import and export filters/transformers of the devices, and also how the preferred routes are chosen at a given device. In our formulation, each problem instance also incorporates a specification of the possible failures and their impact on the routing solutions.

Formally, an SPPF is a tuple with six components:

1. A graph $G = \langle V, E \rangle$ denoting the network topology.
2. A set of "attributes" (*i.e.*, routing messages) $A_\infty = A \cup \{\infty\}$ that may be exchanged between devices. The symbol ∞ represents the absence of a route.
3. A destination $d \in V$ and its initial route announcement $a_d \in A$. For simplicity, each SPPF has exactly one destination (d). (To model a network with many destinations, one would use a set of SPPFs.)
4. A partial order $\prec \subseteq A_\infty \times A_\infty$ ranks attributes. If $a \prec b$ then we say route a is preferred over route b. Any route $a \in A$ is preferred to no route ($a \prec \infty$). This relation determines the way devices select routes along which to forward traffic—every device will always select its most preferred route.[5]
5. A function $\mathsf{trans} : E \to A_\infty \to A_\infty$ that denotes how messages are processed across edges. This function models the route maps and filters that transform route announcements as they enter or leave routers.
6. A bound k on the maximum number of link failures that may occur.

Examples: By choosing an appropriate set of routing attributes, a preference relation and a transfer function, one can model the semantics of commonly used routing protocols. For instance, the Routing Information Protocol (RIP) is a simple shortest paths protocol. It can be modelled by an SPPF where (1) the set of attributes A is the set of integers between 0 and 15 (*i.e.*, the set of permitted path lengths), (2) the preference relation is integer inequality so shorter paths are preferred, and (3) the transfer function increments the received attribute by 1 or drops the route if it exceeds the maximum hop count of 15:

$$\mathsf{trans}(e, a) = \begin{cases} \infty & \text{if } a \geq 15 \\ a + 1 & \text{otherwise} \end{cases}$$

Going beyond simple shortest paths, BGP is a complex, policy-driven protocol that drives the Internet, and increasingly, data centers [19]. Operators often choose BGP due to its high expressiveness. We can model a version of BGP (simplified for presentation) using messages consisting of triples $(\mathrm{LP}, \mathrm{Comm}, \mathrm{Path})$ where LP is an integer-valued local preference, Comm is a set of community values (which are essentially string tags) and Path is a list of nodes, representing

[5] Though there is just one preference relation, when modelling a network, one can include the name of the current router (or the entire path) in each attribute. As a consequence, different routers can have different preferences and make different routing decisions.

the path a routing message has traversed. The transfer function always adds the current device to the Path (or drops the message if a loop is detected) and will modify the LP and Comm components of the attribute according to the device configuration. For instance, one device may attach a community tag to a route and another device may filter or modify routes that have the tag attached.

$$\mathsf{trans}((u,v),a) = \begin{cases} \infty & \text{if } u \in \text{path} \\ \text{config}_{(u,v)}(a) & \text{otherwise} \end{cases} \quad \textbf{where } a = (\text{lp}, \text{comms}, \text{path})$$

The protocol semantics dictates the preference relation (preferring routes with higher local preference first, and shorter paths second). A more complete BGP model is not fundamentally harder to model—it simply has additional attribute fields and more complex transfer and preference relations [21].

SPPF Solutions: In a network, routers will repeatedly exchange messages, applying their transfer functions to neighbor routes and selecting a current best route based on the preference relation, until the network reaches a fixpoint (stable state). Interestingly, Griffin *et al.* [17] showed that all routing solutions can be described via a set of local stability constraints. We exploit this insight to define a series of logical constraints that capture all possible routing behaviors in a setting that includes link failures. More specifically, we define a *solution* (*aka*, *stable state*) $\mathcal{S}$ of an SPPF to be a pair $\langle \mathcal{L}, \mathcal{F} \rangle$ of a labelling $\mathcal{L}$ and a failure scenario $\mathcal{F}$. The labelling $\mathcal{L}$ is an assignment of the final attributes to nodes in the network. If an attribute a is assigned to node v, we say that node has selected (or prefers) that attribute over other attributes available to it. The chosen route also determines packet forwarding. If a node X selects a route from neighbor Y, then X will forward packets to Y. The failure scenario $\mathcal{F}$ is an assignment of 0 (has not failed) or 1 (has failed) to each edge in the network.

A solution $\mathcal{S} = \langle \mathcal{L}, F \rangle$ to an $\mathsf{SPPF} = (G, A, a_\text{d}, \prec, \mathsf{trans}, k)$ is a stable state satisfying the following conditions:

$$\mathcal{L}(u) = \begin{cases} a_\text{d} & u = d \\ \infty & \mathsf{choices}_\mathcal{S}(u) = \emptyset \\ min_\prec(\{a \mid (e,a) \in \mathsf{choices}_\mathcal{S}(u)\}) & \mathsf{choices}_\mathcal{S}(u) \neq \emptyset \end{cases}$$

$$\textbf{subject to} \quad \sum_{e \in E} \mathcal{F}(e) \leq k$$

where the choices from the neighbors of node u are defined as:

$$\mathsf{choices}_\mathcal{S}(u) = \{(e,a) \mid e = \langle u,v \rangle,\ a = \mathsf{trans}(e, \mathcal{L}(v)),\ a \neq \infty,\ \mathcal{F}(e) = 0\}$$

The constraints require that every node has selected the best attribute (according to its preference relation) amongst those available from its neighbors. The destination's label must always be the initial attribute a_d. For verification, this attribute (or parts of it) may be symbolic, which helps model potentially unknown routing announcements from peers outside our network. For other nodes u, the selected attribute a is the minimal attribute from the *choices* available to u. Intuitively, to find the choices available to u, we consider the attributes b chosen by neighbors v of u. Then, if the edge between v and u is not failed, we push b along that edge, modifying it according to the trans function. Finally, failure scenarios are constrained so that the sum of the failures is at most k.

4 Network approximation theory

Given a concrete SPPF and an abstract $\widehat{\mathsf{SPPF}}$, a network abstraction is a pair of functions (f, h) that relate the two. The topology abstraction $f : V \to \widehat{V}$ maps each node in the concrete network to a node in the abstract network, while the attribute abstraction $h : A_\infty \to \widehat{A}_\infty$ maps a concrete attribute to an abstract attribute. The latter allows us to relate networks running protocols where nodes may appear in the attributes (*e.g.* as in the Path component of BGP).

The goal of Origami is to compute compact $\widehat{\mathsf{SPPFs}}$ that may be used for verification. These compact $\widehat{\mathsf{SPPFs}}$ must be closely related to their concrete counterparts. Otherwise, properties verified on the compact $\widehat{\mathsf{SPPF}}$ will not be true of their concrete counterpart. Section 4.1 defines *label approximation*, which provides an intuitive, high-level, semantic relationship between abstract and concrete networks. We also explain some of the consequences of this definition and its limitations. Unfortunately, while this broad definition serves as an important theoretical objective, it is difficult to use directly in an efficient algorithm. Section 4.2 continues our development by explaining two *well-formedness* requirements of network policies that play a key role in establishing label approximation *indirectly*. Finally, Section 4.3 defines *effective SPPF approximation* for well-formed SPPFs. This definition is more conservative than label approximation, but has the advantage that it is easier to work with algorithmically and, moreover, it implies label approximation. We prove the soundness of effective SPPF approximation relative to SPPF approximation proper. See the appendix [**?**] for proofs.

4.1 Label approximation

Intuitively, we say the abstract $\widehat{\mathsf{SPPF}}$ label-approximates the concrete SPPF when SPPF has at least as good a route at every node as $\widehat{\mathsf{SPPF}}$ does.

Definition 1 (Label Approximation). *Consider any solutions $\mathcal{S}$ to SPPF and $\widehat{\mathcal{S}}$ to $\widehat{\mathsf{SPPF}}$ and their respective labelling components $\mathcal{L}$ and $\widehat{\mathcal{L}}$. We say $\widehat{\mathsf{SPPF}}$ label-approximates SPPF when $\forall u \in V.\ h(\mathcal{L}(u)) \preceq \widehat{\mathcal{L}}(f(u))$*

If we can establish a label approximation relation between a concrete and an abstract network, we can typically verify a number of properties of the abstract network and be sure they hold of the concrete network. However, the details of exactly which properties we can verify depend on the specifics of the preference relation ($\prec$). For example, in an OSPF network, preference is determined by weighted path length. Therefore, if we know an abstract node has a path of weighted length n, we know that its concrete counterparts have paths of weighted length of at most n. More importantly, since "no route" is the worst route, we

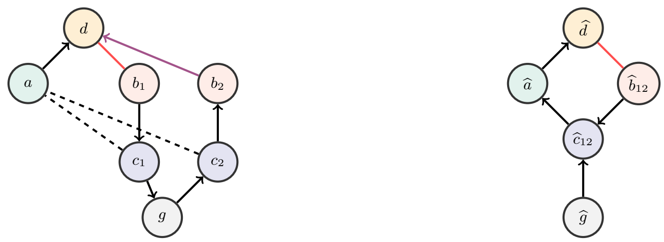


Fig. 2: Concrete network (left) and its corresponding abstraction (right). Nodes c_1, c_2 prefer to route through b_1 (resp. b_2), or g over a. Node b_1 (resp. b_2) drops routing messages that have traversed b_2 (resp. b_1). Red lines indicate a failed link. Dotted lines show a topologically available but unused link. A purple arrow show a route unuseable by traffic from b_1.

know that if a node has any route to the destination in the abstract network, so do its concrete counterparts.

As another example, one may wish to check that a BGP network implements the Gao-Rexford conditions [13], a set of guidelines to prevent inter-domain oscillation. Hence the network should prefer *customer* over *peer* over *provider* (which are three different types of business relationships). Here, we might learn that a particular node routes to a peer in the abstract network and hence we would be guaranteed they route either to a peer or to customer but not to a provider. Notice, however, that in this case, the preference relation (customer $\prec$ peer $\prec$ provider) is determined by the configuration and the configuration could be incorrect — indeed, our goal is to check such configurations. So it would not be wise to count on the correctness of the definition of preference everywhere in the network. In such a situation, one could check the preference relation at every node (a simple, inexpensive local check).

Limitations. Some properties are beyond the scope of our tool (independent of the preference relation). For example, our model cannot reason about quantitative properties such as bandwidth, probability of congestion, or latency.

4.2 Well-formed SPPFs

Not all SPPFs are well-behaved. For example, some never converge and others do not provide sensible models of any real network. To avoid dealing with such poorly-behaved models, we demand henceforth that all SPPFs are *well-formed*. Well-formedness entails that an SPPF is strictly monotonic and isotonic:

$$\forall a, e.\ \ a \neq \infty \Rightarrow a \prec \mathsf{trans}(e, a) \qquad \textit{strict monotonicity}$$

$$\forall a, b, e.\ \ a \preceq b \ \Rightarrow\ \mathsf{trans}(e, a) \preceq \mathsf{trans}(e, b) \qquad \textit{isotonicity}$$

Monotonicity and isotonicity properties are often cited [7,8] as desirable properties of routing policies because they guarantee network convergence and prevent persistent oscillation. In practice too, prior studies have revealed that almost all real network configurations have these properties [14,20].

In our case, these properties help establish additional invariants that tie the routing behavior of concrete and abstract networks together. To gain some

Fig. 3: Concrete network (left) and its corresponding $\forall\exists-\textsf{abstraction}$ (right). Node c prefers to route through x_1, x_2 and nodes x_1, x_2 prefer to route through c.

intuition as to why, consider the networks of Figure 2. The concrete network on the left runs BGP with the routing policy that node c_1 (and c_2) prefers to route through node g instead of a, and that b_1 drops announcements coming from b_2. In this scenario, the similarly configured abstract node $\widehat{b}_{12}$ can reach the destination—it simply takes a route that happens to be less preferred by $\hat{c}_{12}$ than it would if there had been no failure. However, in the concrete analogue, b_1, is *unable* to reach the destination because c_1 only sends it the route through b_2, which it cannot use. In this case, the concrete network has more topological paths than the abstract network, but, counterintuitively, due to the network's routing policy, this turns out to be a disadvantage. Hence having more paths does not necessarily make nodes more accessible. As a consequence, in general, abstract networks cannot soundly overapproximate the number of failures in a concrete network—an important property for the soundness of our theory.

The underlying issue here is that the networks of Figure 2 are not isotonic: suppose $\mathcal{L}'(c_1)$ is the route from c_1 to the destination through node a, we have that $\mathcal{L}(c_1) \prec \mathcal{L}'(c_1)$ but since the transfer function over $\langle b_1, c_1\rangle$ drops routes that have traversed node b_2, we have that $\textsf{trans}(\langle b_1, c_1\rangle, \mathcal{L}(c_1)) \not\prec \textsf{trans}(\langle b_1, c_1\rangle, \mathcal{L}'(c_1))$. Notice that $\mathcal{L}'(c_1)$ is essentially the route that the abstract network uses *i.e.* $h(\mathcal{L}'(c_1)) = \widehat{\mathcal{L}}(\hat{c}_{12})$, hence the formula above implies that $h(\mathcal{L}(b_1)) \not\prec \widehat{\mathcal{L}}(\hat{b}_{12})$ which violates the notion of label approximation.

A similar problem arises when the routing policy is not monotonic. Consider the topology of Figure 3 running eBGP with a routing policy that dictates that node c prefers to route through nodes x_1, x_2 over b and nodes x_1, x_2 prefer to route through c. The abstract node $\widehat{x}_{12}$ has a (unique) choice to route through $\widehat{c}$. On the other hand, depending on how the concrete network convergences, x_2 may route directly to d and c may route through x_2. In this scenario x_2 ends up having a route to the destination that is worse than the one of its abstract counterpart $\widehat{x}_{12}$.

Fortunately, if a network is strictly monotonic and isotonic, such situations never arise. Moreover, we check these properties via an SMT solver using a local and efficient test.

4.3 Effective SPPF approximation

We seek abstract networks that label-approximate given concrete networks. Unfortunately, to directly check that a particular abstract network label approximates a concrete network one must effectively compute their solutions. Doing so would defeat the entire purpose of abstraction, which seeks to analyze large concrete networks *without the expense of computing their solutions directly.*

In order to turn approximation into a useful computational tool, we define *effective approximation*, a set of simple conditions on the abstraction functions f and h that are *local* and can be checked efficiently. When true those conditions imply label approximation. Intuitively effective approximations impose three main restrictions on the abstraction functions [6]:

1. The topology abstraction conforms to the $\forall\exists-\mathsf{abstraction}$ condition; this requires that there is an abstract edge $(\widehat{u}, \widehat{v})$ iff for every concrete node u such that $f(u) = \widehat{u}$ there is some node v such that $f(v) = \widehat{v}$ and $(u, v) \in E$.
2. The abstraction preserves the rank of attributes *(rank-equivalence)*:
$$\forall a, b.\ a \prec b \iff h(a) \mathrel{\widehat{\prec}} h(b)$$
3. The transfer function and the abstraction functions commute *(trans-equivalence)*:
$$\forall e, a.\ h(\mathsf{trans}(e, a)) = \widehat{\mathsf{trans}}(f(e), h(a))$$

Relating abstract and concrete choices: To relate the solutions of the two networks, we must relate the routing choices between abstract and concrete nodes. The examples of section 4.2 demonstrate that, in general, the solutions of the concrete network and the solutions of its effective abstractions are not necessarily label-approximate. At the same time, they provided us with useful insights about the class of networks for which effective abstractions imply label-approximate.

Definition 2. *Given an SPPF and its abstraction* $\widehat{\mathsf{SPPF}}$ *defined by* (f, h) *we say that the two networks are* choice approximate *if their solutions satisfy the following property:*

$$\begin{aligned}&\forall \langle u, v\rangle \in E, \widehat{a} \in \widehat{A}.\ (f(\langle u, v\rangle), \widehat{a}) \in min(\mathsf{choices}_{\widehat{\mathcal{S}}}(f(u))) \implies \\ &\quad (\exists a.\ h(a) \mathrel{\widehat{\preceq}} \widehat{a} \ \wedge\ (\langle u, v\rangle, a) \in \mathsf{choices}_{\mathcal{S}}(u)) \vee \\ &\quad (u \in \mathcal{L}(v).\mathrm{path})\end{aligned}$$

Our initial intuition is that if an abstract node $f(u)$ has an optimal routing choice through the link $\langle f(u), f(v)\rangle$ then the concrete node u has a routing choice over $\langle u, v\rangle$ that is at least as good. Formally this means that $\mathsf{trans}(\langle u, v\rangle, \mathcal{L}(v)) \preceq \mathsf{trans}(\langle f(u), f(v)\rangle, \widehat{\mathcal{L}}(f(v))$. Thinking inductively, one may wonder: *given that v and $f(v)$ are label-approximate, can we prove the same of u and f(u)?* As we saw in section 4.2 the transfer function must be isotonic for this to hold.

The example of Figure 3 illustrated the case where using protocols that have a loop detection mechanism, such as BGP, a concrete node may not have a

[6] See the technical appendix for a full definition

choice that its abstract counterpart had. This is captured in Definition 4 by the case where $u \in \mathcal{L}(v).\text{path}$. To establish label-approximate in this case we need to resort to monotonicity. To understand how monotonicity is used in this case, imagine there was no loop-detection (i.e. the route causing a loop is not dropped), then according to monotonicity, the routing choice v would offer to u would be strictly worse, as v's route uses u and every application of the transfer function produces a worse attribute.

We prove that when these conditions hold, we can approximate any solution of the concrete network with a solution of the abstract network.

Theorem 1. *Given a well-formed* SPPF *and its effective approximation* $\widehat{\mathsf{SPPF}}$*, for any solution* $\mathcal{S} \in \mathsf{SPPF}$ *there exists a solution* $\widehat{\mathcal{S}} \in \widehat{\mathsf{SPPF}}$*, such that their labelling functions are label approximate.*

5 The verification procedure

The first step of verification is to compute a small abstract network that satisfies our SPPF *effective approximation* conditions. We do so by grouping network nodes and edges with equivalent policy and checking the forall-exists topological condition, using an algorithm reminiscent of earlier work [3]. Typically, however, this minimal abstraction will not contain enough paths to prove any fault-tolerance property. To identify a finer abstraction for which we can prove a fault-tolerance property we repeatedly:

1. Search the set of candidate refinements for the smallest *plausible* abstraction.
2. If the candidate abstraction satisfies the desired property, terminate the procedure. (We have successfully verified our concrete network.)
3. If not, examine whether the returned counterexample is an actual counterexample. We do so, by computing the number of concrete failures and check that it does not exceed the desired bound of link failures. (If so, we have found a property violation.)
4. If not, use the counterexample to *learn* how to expand the abstract network into a larger abstraction and repeat.

Both the search for plausible candidates and the way we learn a new abstraction to continue the counterexample-guided loop are explained below.

5.1 Searching for plausible candidates

Though we might know an abstraction is not sufficient to verify a given fault tolerance property, there are many possible refinements. Consider, for example, Figure 4(a) presents a simple concrete network that will tolerate a single link failure, and Figure 4(b) presents an initial abstraction. The initial abstraction will not tolerate any link failure, so we must refine the network. To do so, we choose an abstract node to divide into two abstract nodes for the next iteration. We must also decide which concrete nodes correspond to each abstract node. For

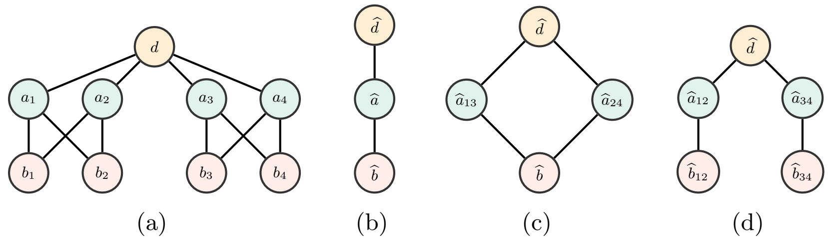


Fig. 4: Eight nodes in (a) are represented using two nodes in the abstract network (b). Pictures (c) and (d) show two possible ways to refine the abstract network (b).

example, in Figure 4(c), node $\hat{a}$ has been split into $\hat{a}_{13}$ and $\hat{a}_{24}$. The subscripts indicate the assignment of concrete nodes to abstract ones.

A significant complication is that once we have generated a new abstraction, we must check that it continues to satisfy the effective approximation conditions, and if not, we must do more work. Figure 4 (c) satisfies those conditions, but if we were to split $\hat{a}$ into $\hat{a}_{12}$ and $\hat{a}_{34}$ rather than $\hat{a}_{13}$ and $\hat{a}_{24}$, the forall-exists condition would be violated—some of the concrete nodes associated with $\hat{b}$ are connected to the concrete nodes in $\hat{a}_{12}$ but not to the ones in $\hat{a}_{34}$ and vice versa. To repair the violation of the forall-exists condition, we need to split additional nodes. In this case, the $\hat{b}$ node, giving rise to diagram (4d).

Overall, the process of splitting nodes and then recursively splitting further nodes to repair the forall-exists condition generates many possible candidate abstractions to consider. A key question is which candidate should we select to proceed with the abstraction refinement algorithm?

One consideration is size: A smaller abstraction avoids taxing the verifier, which is the ultimate goal. However, there are many small abstractions that we can quickly dismiss. Technically, we say an abstraction is *plausible* if all nodes of interest have at least $k+1$ paths to the destination. Implausible abstractions cause nodes to become unreachable with k failures. To check whether an abstraction is plausible, we compute the *min-cut* of the graph. Figure 4(d) is an example of an implausible abstraction that arose after a poorly-chosen split of node $\hat{a}$. In this case, no node has 2 or more paths to the destination and hence they might not be able to reach the destination when there is a failure.

Clearly verification using an implausible abstraction will fail. Instead of considering such abstractions as candidates for running verification on, the refinement algorithm tries refining them further. A key decision the algorithm needs to make when refining an abstraction is *which abstract node to split.* For instance, the optimal refinement of Figure 4(b) is Figure 4(c). If we were to split node $\hat{b}$ instead of $\hat{a}$ we would end up with a sub-optimal (in terms of size) abstraction. Intuitively, splitting a node that lies on the min-cut and can reach the destination (e.g. $\hat{a}$) will increase the number of paths that its neighbors on the unreachable part of the min-cut (e.g. $\hat{b}$) can use to reach the destination.

To summarize, the search for new candidate abstractions involves (1) splitting nodes in the initial abstraction, (2) repairing the abstraction to ensure the forall-exists condition holds, (3) checking that the generated abstraction is *plausible*, and if not, (4) splitting additional nodes on the min cut. This iterative process will often generate many candidates. The *breadth* parameter of the search bounds the total number of plausible candidates we will generate in between verification efforts. Of all the plausible candidates generated, we choose the smallest one to verify using the SMT solver.

5.2 Learning from counterexamples

Any nodes of an abstraction that have a min cut of less than $k+1$ definitely cannot tolerate k faults. If an abstraction is plausible, it satisfies a *necessary* condition for source-destination connectivity, but not a *sufficient* one—misconfigured routing policy can still cause nodes to be unreachable by modifying and/or subsequently dropping routing messages. For instance, the abstract network of Figure 4c is plausible for one failure, but if $\widehat{b}$'s routing policy blocks routes of either $\widehat{a}_{13}$ or $\widehat{a}_{24}$ then the abstract network will not be 1-fault tolerant. Indeed, it is the complexity of routing policy that necessitates a heavy-weight verification procedure in the first place, rather than a simpler graph algorithm alone.

In a plausible abstraction, if the verifier computes a solution to the network that violates the desired fault-tolerance property, some node could not reach the destination because one or more of their paths to the destination could not be used to route traffic. We use the generated counterexample to learn edges that could not be used to route traffic due to the policy on them. To do so, we inspect the computed solution to find nodes $\widehat{u}$ that (1) lack a route to the destination (*i.e.* $\widehat{\mathcal{L}}(\widehat{u}) = \infty$), (2) have a neighbor $\widehat{v}$ that has a valid route to the destination, and (3) the link between $\widehat{u}$ and $\widehat{v}$ is not failed. These conditions imply the absence of a valid route to the destination not because link failures disabled all paths to the destination, but because the network policy dropped some routes. For example, in picture Figure 4c, consider the case where $\widehat{b}$ does not advertise routes from $\widehat{a}_{13}$ and $\widehat{a}_{24}$; if the link between $\widehat{a}_{13}$ and $\widehat{d}$ fails, then $\widehat{a}_{13}$ has no route the destination and we learn that the edge $\langle \widehat{b}, \widehat{a}_{13} \rangle$ cannot be used. In fact, since $\widehat{a}_{13}$ and $\widehat{a}_{12}$ belonged to the same abstract group $\widehat{a}$ before we split them, their routing policies are equal modulo the abstraction function by trans-equivalence. Hence, we can infer that in a symmetric scenario, the link $\langle \widehat{b}, \widehat{a}_{24} \rangle$ will also be unusable.

Refining using learned paths:

Given a set of unuseable edges, learned from a counterexample, we restrict the min cut problems that define the plausible abstractions, by disallowing the use of those edges. Essentially, we enrich the refinement algorithm's topological based analysis (based on min-cut) with knowledge about the policy; the algorithm will have to generate abstractions that are plausible without using those edges. With those edges disabled, the refinement process continues as before.

6 Implementation

Origami uses the Batfish network analysis framework [12] to parse network configurations, and then translate them into a pure functional intermediate representation (IR) designed for network verification. This IR represents the structure of routing messages and the semantics of transfer and preference relations using standard functional data structures.

The translation generates a separate functional program for each destination subnet. In other words, if a network has 100 top-of-rack switches and each such switch announces the subnets for 30 adjacent hosts, then Origami generates 100 functional programs (*i.e.* problem instances). We separately apply our algorithms to each problem instance, converting the functional program to an SMT formula when necessary according to the algorithm described earlier. Since vendor routing configuration languages have limited expressive power (*e.g.*, no loops or recursion) the translation requires no user-provided invariants. We use Z3 [10] to determine satisfiability of the SMT problems. Solving the problems separately (and in parallel) provides a speedup over solving the routing problem for all destinations simultaneously: The individual problems are specialized to a particular destination. By doing so, opportunities for optimizations that reduce the problem size, such as dead code elimination, arise.

Optimizing refinement: During the course of implementing Origami, we discovered a number of optimizations to the refinement phase.

- If the min-cut between the destination and a vertex u is less than or equal to the desired number of disjoint paths, then we do not need to compute another min-cut for the nodes in the unreachable portion of vertices T; we know nodes in T can be disconnected from the destination. This significantly reduces the number of min-cut computations.
- We stop exploring abstractions that are larger in size than the smallest plausible abstraction computed since the last invocation of the SMT solver.
- We bias our refinement process to explore the smallest abstractions first. When combined the previous optimization, this prunes our search space from some abstractions that were unnecessary large.

Minimizing counterexamples: When the SMT solver returns a counterexample, it often uses the maximum number of failures. This is not surprising as maximizing failures simplifies the SMT problem. Unfortunately, it also confounds our analysis to determine whether a counterexample is real or spurious. For instance, if the failure bound is 2, failing 2 abstract links could result in 4 concrete failures, which exceeds the failure bound, and leads the simple analysis to believe the counterexample is spurious, causing another round of refinement. However, it could also be the case that there exists another failure scenario that results in just 2 concrete failures. This latter scenario might even use just a subset of the links from the initial scenario. If we do not detect such issues, we will slow analysis by performing unnecessary iterations of the refinement loop.

To mitigate the effect of this problem, we *could* ask the solver to minimize the returned counterexample, returning a counterexample that corresponds to

Topo	Con V/E	Fail	Abs V/E	Ratio	Abs Time	SMT Calls	SMT Time
FT20	500/8000	1	9/20	55.5/400	0.1	1	0.1
		3	40/192	12.5/41.67	1.0	2	7.6
		5	96/720	5.20/11.1	2.5	2	248
		10	59/440	8.48/18.18	0.9	-	-
FT40	2000/64000	1	12/28	166.7/2285.7	0.1	1	0.1
		3	45/220	44.4/290.9	33	2	12.3
		5	109/880	18.34/72.72	762.3	2	184.1
SP40	2000/64000	1	13/32	153.8/2000	0.2	1	0.1
		3	39/176	51.3/363.6	30.3	1	2
		5	79/522	25.3/122.6	372.2	1	22
FbFT	744/10880	1	20/66	37.2/164.8	0.1	3	1
		3	57/360	13.05/30.22	1	4	18.3
		5	93/684	8/15.9	408.9	-	-

Fig. 5: Compression results. **Topo**: the network topology. **Con V/E**: Number of nodes/edges of concrete network. **Fail**: Number of failures. **Abs V/E**: Number of nodes/edges of the best abstraction. **Ratio**: Compression ratio (nodes/edges). **Abs Time**: Time taken to find abstractions (sec.). **SMT Calls**: Number of calls to the SMT solver. **SMT Time**: Time taken by the SMT solver (sec.).

the fewest concrete link failures. We could do so by providing the solver with additional constraints specifying the number of concrete links that correspond to each abstract link and then asking the solver to return a counterexample that minimizes this sum of concrete failures. Of course, doing so requires we solve a more expensive optimization problem. Instead, given an initial (possibly spurious counter-example), we simple ask the solver to find a new counterexample that (additionally) satisfies this constraint. If it succeeds, we have found a real counterexample. If it fails, we use it to refine our abstraction.

7 Evaluation

We evaluate Origami on a collection of synthetic data center networks that are using BGP to implement shortest-paths routing policies over common industrial datacenter topologies. Data centers are good fit for our algorithms as they can be very large but are highly symmetrical and designed for fault tolerance. Data center topologies (often called *fattree* topologies) are typically organized in layers, with each layer containing many routers. Each router in a layer is connected to a number of routers in the layer above (and below) it. The precise number of neighbors to which a router is connected, and the pattern of said connections, is part of the topology definition. We focus on two common topologies: fattree topologies used at Google (labelled FT20, FT40 and SP40 below) and a different fattree used at Facebook (labelled FB12). These are relatively large data center topologies ranging from 500 to 2000 nodes and 8000 to 64000 edges.

SP40 uses a pure shortest paths routing policy. For other experiments (FT20, FT40, FB12", we augment shortest paths with additional policy that selectively drops routing announcements, for example disabling "valley routing" in various places which allows up-down-up-down routes through the data centers instead of just up-down routes. The pure shortest paths policy represents a best-case scenario for our technology as it gives rise to perfect symmetry and makes our heuristics especially effective. By adding variations in routing policy, we provide a greater challenge for our tool.

Experiments were done on a Mac with a 4GHz i7 CPU and 16GB memory.

7.1 Compression results

Figure 5 shows the level of compression achieved, along with the required time for compression and verification. In most cases, we achieve a high compression ratio especially in terms of links. This drastically reduces the possible failure combinations for the underlying verification process. The cases of 10 link failures on FT20 and 5 link failures on FbFT demonstrate another aspect of our algorithm. Both topologies cannot sustain that many link failures, *i.e.* some concrete nodes have less than 10 (resp. 5) neighbors. We can determine this as we refine the abstraction; there are (abstract) nodes that do not satisfy the min cut requirement and we cannot refine them further. This constitutes an actual counterexample and explains why the abstraction of FT20 for 10 link failures is smaller than the one for 5 link failures. Importantly, we did not use the SMT solver to find this counterexample. Likewise, we did not need to run a min cut on the much larger concrete topology. Intuitively, the rest of the network remained abstract, while the part that led to the counterexample became fully concrete.

7.2 Verification performance

The verification time of Origami is dominated by abstraction time and SMT time, which can be seen in Figure 5. In practice, there is also some time taken to parse and pre-process the configurations but it is negligible. The abstraction time is highly dependent on the size of the network and the abstraction search breadth used. In this case, the breadth was set to 25, a relatively high value.

While the verification time for a high number of link failures is not negligible, we found that verification without abstraction is essentially impossible. We used Minesweeper[2], the state-of-the-art SMT-based network verifier, to verify the same fault tolerance properties and it was unable to solve any of our queries. This is not surprising, as SMT-based verifiers do not scale to networks beyond the size of FT20 even without any link failures.

7.3 Refinement effectiveness

We now evaluate the effectiveness of our search and refinement techniques.

Effectiveness of search. To assess the effectiveness of the search procedure, we compute an initial abstraction of the FT20 network suitable for 5 link failures, using different values of the search breadth. On top of this, we additionally consider the impact of some of the heuristics described in Section 5. Figure 6

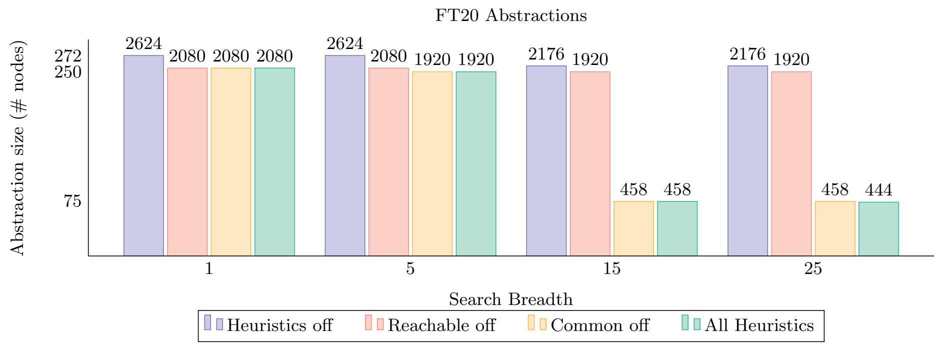


Fig. 6: The initial abstraction of FT20 for 5 link failures using different heuristics and search breadth. On top of the bars is the number of edges of each abstraction.

presents the size (the number of nodes are on the y axis and the number of edges on top of the bars) of the computed abstractions with respect to various values for the breadth of search and sets of heuristics:

- Heuristics off means that (almost) all heuristics are turned off. We still try to split nodes that are on the cut-set.
- Reachable off means that we do not bias towards splitting of nodes in the reachable portion of the cut-set.
- Common off means that we do not bias towards splitting reachable nodes that have the most connections to unreachable nodes.

The results of this experiment show that in order to achieve effective compression ratios we need to employ both smart heuristics and a wide search through the space of abstractions. It is possible that increasing the search breadth would make the heuristics redundant, however, in most cases this would make the refinement process exceed acceptable time limits.

Use of counterexamples. We now assess how important it is to 1) use symmetries in policy to infer more information from counterexamples, and 2) minimize the counterexample provided by the solver.

We see in Figure 7 that disabling them increases number of refinement iterations. While each of these refinements is performed quickly, the same cannot be guaranteed of the verification process that runs between them. Hence, it is important to keep refinement iterations as low as possible.

8 Related work

Our approach to network fault-tolerance verification draws heavily from ideas in prior work exploiting symmetry and abstraction in model checking [18,4,6] and automatic abstraction refinement via CEGAR [1,5,9]. However, we apply these ideas to network routing, which introduces different challenges and opportunities. For example, our notion of abstraction ($\forall\exists-$abstraction) differs from the typical existential abstraction used in model checking [6]. In addition, we have to deal with network topological structure and asymmetries introduced by failures.

Bonsai [3] and Surgeries [23] both leverage abstraction to accelerate verification for routing protocols and packet forwarding respectively. Both tools compute a single abstract network that is bisimilar to the original concrete network. Alas, neither approach can be used to reason about properties when faults may occur.

Minesweeper [2] is a general approach to control plane verification based on a stable state encoding, which leverages an SMT solver in the back-end. It supports a wide range of routing protocols and properties, including fault tolerance properties. Our compression is complementary to such tools; it is used to alleviate the scaling problem that Minesweeper faces with large networks.

With respect to verification of fault tolerance, ARC [14] translates a limited class of routing policies to a weighted graph where fault-tolerance properties can be checked using graph algorithms. However, ARC only handles shortest path routing and cannot support stateful features such as BGP communities, or local preference, etc. While ARC applies graph algorithms on a statically-computed graph, we use graph algorithms as part of a refinement loop in conjunction with a general purpose solver.

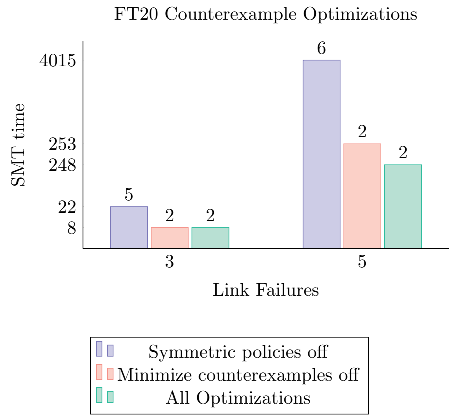


Fig. 7: Effectiveness of minimizing counterexamples and of learning unused edges. On top of the bars is the number of SMT calls. The refinement time is insignificant so we omit it.

9 Conclusions

We present a new theory of distributed routing protocols in the presence of bounded link failures, and we use the theory to develop algorithms for network compression and counterexample-guided verification of fault tolerance properties. In doing so, we observe that (1) even though abstract networks route differently from concrete ones in the presence of failures, the concrete routes wind up being "at least as good" as the abstract ones when networks satisfy reasonable well-formedness constraints, and (2) using efficient graph algorithms (min cut) in the middle of the CEGAR loop speeds the search for refinements.

We implemented our algorithms in a network verification tool called Origami. Evaluation of the tool on synthetic networks shows that our algorithms accelerate verification of fault tolerance properties significantly, making it possible to verify networks out of reach of other state-of-the-art tools.

Future work. Relaxing the strict monotonicity condition is an obvious extension to our theory. In [3], they handle potentially non-monotonic features of BGP (local preference) using a conservative approach that statically bounds the number of abstract nodes required to represent a group of nodes running a non-monotic policy. Whether this approach applies in conjunction with failures needs to be studied.

Concerning the algorithm, it would be interesting to devise refinement algorithms that can efficiently handle properties beyond reachability, for instance (weighted) path length.

References

1. Ball, T., Majumdar, R., Millstein, T.D., Rajamani, S.K.: Automatic predicate abstraction of C programs. In: Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). pp. 203–213 (2001)
2. Beckett, R., Gupta, A., Mahajan, R., Walker, D.: A general approach to network configuration verification. In: SIGCOMM (August 2017)
3. Beckett, R., Gupta, A., Mahajan, R., Walker, D.: Control plane compression. In: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. pp. 476–489. ACM (2018)
4. Clarke, E.M., Filkorn, T., Jha, S.: Exploiting symmetry in temporal logic model checking. In: Computer Aided Verification, 5th International Conference, CAV, Proceedings. pp. 450–462 (1993)
5. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Computer Aided Verification, 12th International Conference, CAV, Proceedings. pp. 154–169 (2000)
6. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. **16**(5), 1512–1542 (September 1994)
7. Daggitt, M.L., Gurney, A.J.T., Griffin, T.G.: Asynchronous convergence of policy-rich distributed bellman-ford routing protocols. pp. 103–116. SIGCOMM (2018)
8. Daggitt, M.L., Gurney, A.J., Griffin, T.G.: Asynchronous convergence of policy-rich distributed bellman-ford routing protocols. In: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. pp. 103–116. ACM (2018)
9. Das, S., Dill, D.L.: Successive approximation of abstract transition relations. In: Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science. pp. 51–. LICS '01 (2001)
10. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS (2008)
11. Feamster, N., Rexford, J.: Network-wide prediction of BGP routes. IEEE/ACM Trans. Networking **15**(2) (2007)
12. Fogel, A., Fung, S., Pedrosa, L., Walraed-Sullivan, M., Govindan, R., Mahajan, R., Millstein, T.: A general approach to network configuration analysis. In: NSDI (2015)
13. Gao, L., Rexford, J.: Stable internet routing without global coordination. In: SIGMETRICS (2000)
14. Gember-Jacobson, A., Viswanathan, R., Akella, A., Mahajan, R.: Fast control plane analysis using an abstract representation. In: SIGCOMM (2016)

15. Gill, P., Jain, N., Nagappan, N.: Understanding network failures in data centers: Measurement, analysis, and implications. In: SIGCOMM (2011)
16. Godfrey, J.: The summer of network misconfigurations. https://blog.algosec.com/2016/08/business-outages-caused-misconfigurations-headline-news-summer.html (2016)
17. Griffin, T.G., Shepherd, F.B., Wilfong, G.: The stable paths problem and interdomain routing. IEEE/ACM Trans. Networking **10**(2) (2002)
18. Kesten, Y., Pnueli, A.: Control and data abstraction: The cornerstones of practical formal verification. Software Tools for Technology Transfer **4**, 2000 (2000)
19. Lapukhov, P., Premji, A., Mitchell, J.: Use of BGP for routing in large-scale data centers. Internet draft (2015)
20. Lopes, N.P., Rybalchenko, A.: Fast bgp simulation of large datacenters. In: International Conference on Verification, Model Checking, and Abstract Interpretation. pp. 386–408. Springer (2019)
21. Lougheed, K.: A border gateway protocol (bgp). RFC 1163, RFC Editor (1989), http://www.rfc-editor.org/rfc/rfc1163.txt, http://www.rfc-editor.org/rfc/rfc1163.txt
22. Mahajan, R., Wetherall, D., Anderson, T.: Understanding BGP misconfiguration. In: SIGCOMM (2002)
23. Plotkin, G.D., Bjørner, N., Lopes, N.P., Rybalchenko, A., Varghese, G.: Scaling network verification using symmetry and surgery. In: POPL (2016)
24. Quoitin, B., Uhlig, S.: Modeling the routing of an autonomous system with c-bgp. Netwrk. Mag. of Global Internetwkg. **19**(6), 12–19 (November 2005)
25. Sobrinho, J.a.L.: An algebraic theory of dynamic network routing. IEEE/ACM Trans. Netw. **13**(5), 1160–1173 (October 2005)
26. Weitz, K., Woos, D., Torlak, E., Ernst, M.D., Krishnamurthy, A., Tatlock, Z.: Formal semantics and automated verification for the border gateway protocol. In: NetPL (2016)

Appendix

A1: Summary of SPPF Definitions

SPPF instance $\boxed{\mathsf{SPPF} = (G, A, a_{\mathrm{d}}, \prec, \mathsf{trans}, k)}$

G	$= (V, E, d)$	*network topology*
V		*topology vertices*
E	$: V \times V$	*topology edges*
d	$: V$	*destination vertex*
A_∞	$= A \cup \{\infty\}$	*routing attributes*
a_{d}	$: A_\infty$	*initial route*
$\prec$	$\subseteq A_\infty \times A_\infty$	*comparison relation*
trans	$: E \times A_\infty \to A_\infty$	*transfer function*
k		*number of link failures*

Well-formed SPPF

$\forall v.\ \ (v, v) \notin E$	*self-loop freedom*
$\forall e.\ \ \mathsf{trans}(e, \infty) = \infty$	*non-spontaneity*
$\forall a, e.\ \ a \neq \infty \Rightarrow a \prec \mathsf{trans}(e, a)$	*strict monotonicity*
$\forall a, b, e.\ \ a \preceq b \Rightarrow \mathsf{trans}(e, a) \preceq \mathsf{trans}(e, b)$	*isotonicity*
$\forall a, b, u, v.\ a \preceq b\ \wedge u \notin a.\text{path} \Rightarrow\ \mathsf{trans}(\langle u, v\rangle, a) \preceq \mathsf{trans}(\langle u, v\rangle, b)$	*bgp-isotonicity*

SPPF solution $\boxed{\mathcal{S} \triangleq \langle \mathcal{L}, \mathcal{F} \rangle : (V \to A) \times (E \to \{0, 1\})}$

$$\sum_{e \in E} \mathcal{F}(e) \leq k$$

$$\mathcal{L}(u) = \begin{cases} a_{\mathrm{d}} & u = d \\ \infty & \mathsf{choices}_{\mathcal{S}}(u) = \emptyset \\ min_{\prec}(\{a \mid (e, a) \in \mathsf{choices}_{\mathcal{S}}(u)\}) & \mathsf{choices}_{\mathcal{S}}(u) \neq \emptyset \end{cases}$$

where $\mathsf{choices}_{\mathcal{S}}(u) = \{(e, a) \mid e = \langle u, v\rangle,\ a = \mathsf{trans}(e, \mathcal{L}(v)), a \neq \infty, \mathcal{F}(e) = 0\}$

Effective approximations $\boxed{(f, h) : (V \to \widehat{V}) \times (A \to \widehat{A})}$

$(f(d) = \widehat{d}) \wedge (\forall d'.\ d \neq d' \implies f(d') \neq \widehat{d})$	*dest-equivalence*
$h(\infty) = \infty$	*drop-equivalence*
$h(a_d) = \widehat{a_d}$	*orig-equivalence*
$\forall a, b.\ a \prec b \iff h(a)\ \widehat{\prec}\ h(b)$	*rank-equivalence*
$\forall e, a.\ h(\mathsf{trans}(e, a)) = \widehat{\mathsf{trans}}(f(e), h(a))$	*trans-equivalence*
$\forall e, a.\ e = (u, v)\ \wedge\ f(u) \notin h(a).\text{path} \implies h(\mathsf{trans}(e, a)) = \widehat{\mathsf{trans}}(f(e), h(a))$	*transfer-equivalence-bgp*
$(\widehat{u}, \widehat{v}) \in \widehat{E} \iff \big(\forall u.\ f(u) = \widehat{u} \implies \exists v.\ f(v) = \widehat{v} \wedge (u, v) \in E\big)$	$\forall\exists-$abstraction

Definition 3 (Forwarding Relation). *Given an SPPF and the labelling component of its solution we define the edges that are used to forward traffic as follows:*

$$\mathsf{fwd}_{\mathcal{S}}(u) = \{e \mid (e, a) \in \mathsf{choices}_{\mathcal{S}}(u), a \approx \mathcal{L}(u)\}$$

where $a_1 \approx a_2 \triangleq a_1 \not\prec a_2 \wedge a_2 \not\prec a_1$

A2: Proofs of Theorems

Our proofs proceed by induction on the forwarding relation of the abstract network; hence we first establish that forwarding is a well-founded relation.

Lemma 1. *Let S be a finite set and $\sqsubset$ a relation on S. If its transitive closure is irreflexive then $\sqsubset$ is well-founded.*

Proof. Suppose there is an infinite descending chain. As the set S is finite, this implies that the chain has repeating elements, i.e.

$$\dots a_j \sqsubset \dots \sqsubset a_j \dots$$

But this implies that $a_j \sqsubset^+ a_j$, a contradiction. Hence there are no infinite descending chains and $\sqsubset$ is well-founded.

Lemma 2. *The relation $b \sqsubset a = f(a, b) \in \mathsf{fwd}_{\widehat{\mathcal{S}}}(f(a))$ over the set of concrete vertices V is well-founded.*

Proof. First of all, notice that all routing protocols under consideration are loop-free. Hence the transitive closure of $\mathsf{fwd}_{\widehat{\mathcal{S}}}$ is irreflexive. Moreover, the set of concrete vertices V is finite, hence by Lemma 1 $\sqsubset$ is well-founded.

Notation 1 (Min) *We write $min(A)$ to denote an element $a \in A$ s.t $\forall b \in A.\ a \preceq b$.*

A note about BGP. BGP has an implicit loop detection mechanism which drops a route when a loop is formed. Intuitively, we can model this mechanism using a transfer function of the following form:

$$\mathsf{trans}(\langle u, v \rangle, a) = \begin{cases} \infty & \text{if } u \in \text{path} \\ \text{config}_{(u,v)}(a) & \text{otherwise} \end{cases}$$
$$\textbf{where } a = (\text{lp}, \text{comms}, \text{path})$$

where config depends on the specific configuration of the device. To support this idiom we have to adopt many of our definitions to BGP friendly variants.

In our proofs, we do not explicitly distinguish the cases between BGP and non-BGP protocols, but we always prove the extra preconditions to use any BGP variants of our definitions. For protocols which do not have utilize this explicit loop-detection mechanism, we can simple omit/ignore the proof of these preconditions.

In order to relate the labelling functions of two networks we must relate their ($\prec$-minimal) choices.

Definition 4. *Given an SPPF and its abstraction $\widehat{\mathsf{SPPF}}$ defined by (f, h) we say that the two networks are* choice approximate *if their solutions satisfy the following property:*

$$\begin{aligned}&\forall \langle u, v\rangle \in E, \widehat{a} \in \widehat{A}.\ (f(\langle u, v\rangle), \widehat{a}) \in min(\mathsf{choices}_{\widehat{\mathcal{S}}}(f(u))) \implies \\ &\quad (\exists a.\ h(a) \mathrel{\widehat{\preceq}} \widehat{a} \ \wedge\ (\langle u, v\rangle, a) \in \mathsf{choices}_{\mathcal{S}}(u)) \vee \\ &\quad (u \in \mathcal{L}(v).\text{path})\end{aligned}$$

Intuitively, every node in the concrete network will have a choice over some edge that is at least as good as the choice of its abstraction over the corresponding abstract edge. A corner case is when this choice creates a loop in the concrete network; but in this case it must be that the concrete network already has a better choice (by monotonicity).

Next, we prove that choice approximate implies label approximate.

Lemma 3. *Given an SPPF and its effective approximation $\widehat{\mathsf{SPPF}}$ defined by (f, h), if the networks are choice approximate then they are label approximate.*

Proof. Consider a node $u \in V$. We proceed by well-founded induction on $\sqsubset$. We distinguish two cases for u:

- ($\mathbf{u} = \mathbf{d}$) By definition $\mathcal{L}(d) = a_d$ and $\widehat{\mathcal{L}}(\widehat{d}) = \widehat{a_d}$, and using *orig-equivalence* we get that $\mathcal{L}(d) = \widehat{\mathcal{L}}(\widehat{d})$.
- ($\mathbf{u} \neq \mathbf{d}$) For vertices other than the destination there are two more cases:
  - ($\widehat{\mathcal{L}}(f(u)) = \infty$) then the goal holds trivially as $\forall \widehat{a}.\ \widehat{a} \mathrel{\widehat{\preceq}} \infty$.
  - ($\widehat{\mathcal{L}}(f(u)) \neq \infty$) then by definition there exists some vertex $\widehat{v}$ s.t. $\langle f(u), \widehat{v}\rangle \in \mathsf{fwd}_{\widehat{\mathcal{S}}}(f(u))$, i.e. $f(u)$ has a $\widehat{\prec}$-minimal choice which it forwards to:

    $$(\langle f(u), \widehat{v}\rangle, \widehat{a}) \in min(\mathsf{choices}_{\widehat{\mathcal{S}}}(f(u)))$$

    for $\widehat{a} = \widehat{\mathsf{trans}}(\langle f(u), \widehat{v}\rangle, \widehat{\mathcal{L}}(\widehat{v}))$. Moreover notice that $\langle f(u), \widehat{v}\rangle \in \widehat{E}$ and hence by $\forall\exists{-}\mathsf{abstraction}$ we have that $\exists v \in V.\ f(v) = \widehat{v}$. Using $u, v, \widehat{a}$ we instantiate the choice approximate assumption which leads to two cases:
    1. Either, $\exists a \in A.\ h(a) \mathrel{\widehat{\preceq}} \widehat{a}$ and $(\langle u, v\rangle, a) \in \mathsf{choices}_{\mathcal{S}}(u)$, i.e. vertex u has at least one choice that is as good as the best choice of the abstract vertex $f(u)$, hence either $\mathcal{L}(u) = a$ or there is some better choice and $\mathcal{L}(u) \prec a$. In any case, as $h(a) \mathrel{\widehat{\preceq}} \widehat{a}$ and $\widehat{a} \approx \widehat{\mathcal{L}}(f(u))$ we have that $h(\mathcal{L}(u)) \mathrel{\widehat{\preceq}} \widehat{\mathcal{L}}(f(u))$.
    2. Or $u \in \mathcal{L}(v)$. path, then notice that by monotonicity:

       $$\mathcal{L}(u) \prec \mathcal{L}(v) \tag{1}$$

       and moreover, by $\widehat{\mathcal{L}}(f(u)) = \widehat{\mathsf{trans}}(\langle f(u), f(v)\rangle, \widehat{\mathcal{L}}(f(v))) \neq \infty$ and monotonicity we deduce that:

       $$\widehat{\mathcal{L}}(f(v)) \mathrel{\widehat{\prec}} \widehat{\mathcal{L}}(f(u)) \tag{2}$$

Equation (1) and *rank-equivalence* imply that

$$h(\mathcal{L}(u)) \mathrel{\widehat{\prec}} h(\mathcal{L}(v)) \tag{3}$$

Finally, $v \sqsubset u$ as $\langle f(u), f(v)\rangle \in \mathsf{fwd}_{\widehat{\mathcal{S}}}(f(u))$ and hence we can utilize the induction hypothesis to deduce that

$$h(\mathcal{L}(v)) \mathrel{\widehat{\preceq}} \widehat{\mathcal{L}}(f(v)) \tag{4}$$

By Equation (2), Equation (4) and transitivity we get that $h(\mathcal{L}(v)) \mathrel{\widehat{\prec}} \widehat{\mathcal{L}}(f(u))$ and using Equation (3) and transitivity we conclude that $h(\mathcal{L}(u)) \mathrel{\widehat{\prec}} \widehat{\mathcal{L}}(f(u))$.

Lemma 4 (Surjective attribute abstraction). *For every vertex u there exists some attribute $a \in A$ such that $h(a) = \widehat{\mathcal{L}}(f(u))$.*

Proof. The proof proceeds by well-founded induction on $\sqsubset$.

- $(\mathbf{u} = \mathbf{d})$ then the equation is satisfied by a_d.
- $(\mathbf{u} \neq \mathbf{d})$ then we distinguish two cases:
  - $(\widehat{\mathcal{L}}(f(u)) = \infty)$ then the equation is satisfied by $a = \infty$.
  - $(\widehat{\mathcal{L}}(f(u)) \neq \infty)$ then by definition, there exists some $\widehat{v} \in \widehat{V}$ s.t.:

$$(\langle f(u), \widehat{v}\rangle, \widehat{\mathsf{trans}}(\langle f(u), \widehat{v}\rangle, \widehat{\mathcal{L}}(\widehat{v}))) \in min(\mathsf{choices}_{\widehat{\mathcal{S}}}(f(u))) \tag{5a}$$

$$\langle f(u), \widehat{v}\rangle \in \mathsf{fwd}_{\widehat{\mathcal{S}}}(f(u)) \tag{5b}$$

By $\forall\exists\mathsf{-abstraction}$, $\exists v.\ f(v) = \widehat{v}$ and $\langle u, v\rangle \in E$ and Equation (5b) implies that $v \sqsubset u$ hence by induction hypothesis $\exists a_v.\ h(a_v) = \widehat{\mathcal{L}}(f(v))$. Further notice, that $\langle f(u), f(v)\rangle \notin h(a_v)$.path since $\widehat{\mathcal{L}}(f(u)) \neq \infty$. Hence we can apply transfer-equivalence:

$$\widehat{\mathcal{L}}(f(u)) = \widehat{\mathsf{trans}}(\langle f(u), f(v)\rangle, h(a_v)) \qquad \text{by } \textit{transfer-equivalence}$$
$$\widehat{\mathcal{L}}(f(u)) = h(\mathsf{trans}(\langle u, v\rangle, a_v))$$

Thus we pick $a = \mathsf{trans}(\langle u, v\rangle, a_v)$ as the witness to the existential as it satisfies the equation.

Next we prove that the solutions of the concrete and abstract network will be choice approximate.

Theorem 2. *Given a well-formed $\mathit{SPPF} = (G, A, a_d, \prec, \mathsf{trans}, k)$ and its effective approximation $\widehat{\mathsf{SPPF}} = (\widehat{G}, \widehat{A}, \widehat{a_d}, \widehat{\prec}, \widehat{\mathsf{trans}}, k)$ defined by (f, h), for any solution $(\mathcal{L}, \mathcal{F}) \in \mathit{SPPF}$ and $\widehat{(\mathcal{L}, \mathcal{F})} \in \widehat{\mathsf{SPPF}}$, $\mathcal{L}$ and $\widehat{\mathcal{L}}$ are choice approximate.*

Proof. Consider a vertex $u \in V$. We strengthen our induction hypothesis by proving choice and label approximate at the same time, and proceed by well-founded induction on the $\sqsubset$ relation.

We consider two separate cases for u:

- ($\mathbf{u} = \mathbf{d}$) By definition $\mathcal{L}(d) = a_d$ and $\widehat{\mathcal{L}}(\widehat{d}) = \widehat{a_d}$. Moreover $\widehat{\mathcal{L}}(f(d)) = \widehat{\mathcal{L}}(\widehat{d})$ by *dest-equivalence* and hence $h(\mathcal{L}(d)) \widehat{\preceq} \widehat{\mathcal{L}}(f(d))$ follows trivially by *orig-equivalence*.
- ($\mathbf{u} \neq \mathbf{d}$) We prove choice approximate first. By hypothesis we have that $(\langle f(u), f(v)\rangle,\ \widehat{a}) \in min(\mathsf{choices}_{\widehat{\mathcal{S}}}(f(u)))$, which implies:

$$\langle f(u), f(v)\rangle \in \widehat{E} \tag{6a}$$

$$\widehat{a} = \widehat{\mathsf{trans}}(\langle f(u), f(v)\rangle, \widehat{\mathcal{L}}(f(v))) \tag{6b}$$

$$\widehat{a} \neq \infty \tag{6c}$$

$$\widehat{\mathcal{F}}(\langle f(u), f(v)\rangle) = 0 \tag{6d}$$

$$\langle f(u), f(v)\rangle \in \mathsf{fwd}_{\widehat{\mathcal{S}}}(f(u)) \tag{6e}$$

We distinguish two cases:

- ($\mathbf{u} \notin \mathcal{L}(\mathbf{v})$.path)
  Notice that $\langle u, v\rangle \in E$ by $\forall\exists-$abstraction and $\mathcal{F}(\langle u, v\rangle) = 0$ by definition of $\widehat{\mathcal{F}}$. It suffices to prove that $a = \mathsf{trans}(\langle u, v\rangle, \mathcal{L}(v)) \neq \infty$ and $h(a) \widehat{\preceq} \widehat{a}$. Since $\langle f(u), f(v)\rangle \in \mathsf{fwd}_{\widehat{\mathcal{S}}}(f(u))$ we have that $v \sqsubset u$ and we can apply the induction hypothesis to deduce that $h(\mathcal{L}(v)) \widehat{\preceq} \widehat{\mathcal{L}}(f(v))$. By Lemma 4 we have that:

  $\exists a_v.\ h(a_v) = \widehat{\mathcal{L}}(f(v))$

  $h(\mathcal{L}(v)) \widehat{\preceq} h(a_v)$ — by *induction hypothesis*

  $\mathcal{L}(v) \preceq a_v$ — by *rank-equivalence*

  $\mathsf{trans}(\langle u, v\rangle, \mathcal{L}(v)) \preceq \mathsf{trans}(\langle u, v\rangle, a_v)$ — by *isotonicity as* $u \notin \mathcal{L}(v)$.path

  $h(\mathsf{trans}(\langle u, v\rangle, \mathcal{L}(v) \widehat{\preceq} \widehat{\mathsf{trans}}(\langle f(u), f(v)\rangle, h(a_v))$ — by *transfer-equivalence-bgp as* $f(u) \notin h(a_v)$.path since otherwise $\widehat{a} = \infty$ contradicting Equation (6d)

  $h(a) \widehat{\preceq} \widehat{a}$
- ($\mathbf{u} \in \mathcal{L}(\mathbf{v})$.path) The goal is trivial.

Having established that the two SPPFs are choice-approximate we leverage Lemma 3 to obtain label approximate and conclude the proof.

Definition 5 (Failures approximation). *Given an SPPF and its abstraction, for a selection of concrete link failures* $\mathcal{F} : V \to \{0, 1\}$ *we write* $\widehat{\mathcal{F}} : \widehat{V} \to \{0, 1\}$ *to denote a selection of abstract link failures such that:*

$$\widehat{\mathcal{F}}(\widehat{e}) = 1 \iff \exists e. e \mapsto \widehat{e} \wedge \mathcal{F}(e) = 1$$

Theorem 3. *Given a well-formed* $\mathit{SPPF} = (G, A, a_d, \prec, \mathsf{trans}, k)$ *and its effective approximation* $\widehat{\mathsf{SPPF}} = (\widehat{G}, \widehat{A}, \widehat{a_d}, \widehat{\prec}, \widehat{\mathsf{trans}}, k)$*, for any solution* $\mathcal{S} \in \mathit{SPPF}$ *there exists a solution* $\widehat{\mathcal{S}} \in \widehat{\mathsf{SPPF}}$*, such that their labelling functions are label approximate.*

Proof. Suppose $\mathcal{S} = \langle \mathcal{L}, \mathcal{F} \rangle$. We pick $\widehat{\mathcal{F}}$ as described in Definition 5. It is easy to see that

$$\sum_{e \in E} \mathcal{F}(e) \leq K \implies \sum_{\widehat{e} \in \widehat{E}} \widehat{\mathcal{F}}(\widehat{e}) \leq K$$

Since the transfer function of $\widehat{\mathsf{SPPF}}$ is strictly monotonic and isotonic, we know by [25] that it converges to a unique labelling solution $\widehat{\mathcal{L}}$. By Theorem 2 and Lemma 3 $\mathcal{L}$ and $\widehat{\mathcal{L}}$ are label approximate.